Performance Analysis and Improved Communication Overlap for a Seismic Modeling Application on Large InfiniBand Clusters

Sreeram Potluri*, Sayantan Sur*, Ping Lai*, Karen Tomko†, Yifeng Cui‡ and Dhabaleswar K. Panda*
* Department of Computer Science and Engineering, The Ohio State University
Email: {potluri, surs, laipi, panda}@cse.ohio-state.edu
† Ohio Supercomputer Center
Email: ktomko@osc.edu
‡ San Diego Supercomputer Center
Email: yfcui@sdsc.edu

Abstract*—High-performance scientific applications modeling natural phenomena are pushing the boundaries of modern parallel computing systems. These applications are computation and communication intensive. Continual scaling of these applications to even larger systems is key to gaining critical insights into natural phenomena. At the same time, performance optimization of large real-world scientific applications is becoming increasingly challenging due to increasing complexity of various components, such as computer architecture, message-passing library, network architecture and finally application algorithms. Effective optimization of applications increasingly requires careful understanding of cross-cutting issues. AWM-Olsen is a heavily used NSF Teragrid seismic modeling application which is communication intensive. In a previous paper we presented our design modifications to AWM-Olsen using MPI-2 RMA semantics. In this paper, we present a detailed performance analysis and further optimization techniques for the updated MPI-2 version of AWM-Olsen. We propose an improved design to increase overlap of computation and communication using MVAPICH2, a popular MPI-2 implementation on InfiniBand. Additionally, we propose using application logic aware loop fusion techniques to balance computation load within processes in a parallel job. Using the combination of our proposed optimizations, time spent in communication can be reduced by 72% over the previous MPI-2 version and by 89% over the unmodified MPI-1 version on 8K processes. This results in an overall speedup in the execution time by 8%, and 15% respectively. The experiments were carried out on the Ranger cluster at Texas Advanced Computing Center.

I. Introduction

Computational models provide key insights into complex natural phenomena, such as the seismic waves which lead to earthquakes. In order to gain deeper understanding and accuracy, these models must be run at very high resolution and scale. Although modern computing systems provide increasing aggregate compute power (FLOPS), inherent communication costs in the underlying parallel algorithms usually prevent perfect scaling. In order to scale to next generation systems, it is extremely critical that communication costs be minimized to the extent possible. Typically, scientists have to prove the scalability of the application in order to get access to leading edge systems such as Jaguar [1] at the OLCF and Blue Waters [2] at NCSA. Often, one of the pre-requisites of access to such large scale systems is that the target applications demonstrate effective latency hiding through overlapping computation and communication [3]. This is to ensure that applications that operate on such large scales do not waste valuable system time.

AWM-Olsen [4] is a community model used by researchers at Southern California Earthquake Center (SCEC). It is a heavily used NSF-Teragrid and DOE-INCITE application consuming large amount of computation cycles. AWM-Olsen has been scaled to tens/hundreds of thousands of processor cores of Ranger at TACC, Jaguar and Kraken at OLCF/NICS, and BlueGene machines at IBM and ANL. Some of the most detailed simulations to date of earthquakes along the San Andreas fault were carried out using this code, including the well-known TeraShake, SCEC ShakeOut simulations. Therefore, continued scaling of AWM-Olsen is of great interest to the scientific community. AWM-Olsen is written using Fortran 90 and MPI-1 point-to-point communication semantics. Processes are arranged in a 3D Cartesian grid. AWM-Olsen reports two metrics, the average and maximum times taken. Maximum time is recorded at the process which is last to finish (either due to computation or communication). This is due to the fact that processes inside the 3D cuboid have more neighbors, hence more communication, than those on the boundary. In addition, different computation models are employed for interior and boundary data.

The original AWM-Olsen written using MPI-1 did not effectively overlap Communication and computation. When run on Ranger at TACC [5] on 8K processes with a $128 \times 128 \times 128$ data grid per process, on an average each process spent 17% of its execution time in communication. The process which reported maximum execution time, spent 24% of its time in communication. Recently, we optimized AWM-Olsen by using MPI-2 RMA semantics in [6]. Our evaluation using MVAPICH2 [7], a popular MPI-2 implementation on InfiniBand, showed improved performance at scale. The improved MPI-2 version attempts to overlap communication with computation and is successful to a certain extent. This version improved average and maximum communication times to around 7%, and 20% respectively. The motivation behind the work presented in this paper is to analyze the recently improved MPI-2 version in depth. The aim of our analysis is to uncover performance bottlenecks and explore strategies to further reduce time spent in communication routines by overlapping it with computation. Specifically, we aim to answer the following questions in this paper:

1) What factors do we need to consider in order to increase the percentage of overlapped communication?
2) Can we leverage knowledge of internal MVAPICH2

design and InfiniBand architecture to use communication channels that improve overlap?

3) Can we optimize computation loops in such a way so as to reduce load imbalance and minimize impact of process skew?

To address these questions, we follow a three-pronged approach. First, we analyze the computation pattern in AWM to reveal data dependencies which must be unraveled in order to expose more overlap potential. Secondly, we analyze communication channels in MVAPICH2 to appropriately choose a mode under which more communication can be overlapped, regardless of whether it is inter-node or intra-node. Finally, we employ loop fusion techniques in an *application logic aware* manner to AWM-Olsen. The loop fusion techniques speed up the computation functions and reduces load imbalance and skew among processors, leading to more effective overlap. These design enhancements are explained in greater detail in Section IV. Using the combination of our proposed optimizations, time spent in communication can be reduced by 72% over the previous MPI-2 version and by 89% over the unmodified MPI-1 version on 8K processes. The average and maximum communication times of the optimized code are 2%, and 8.9% of application run-time respectively. The proposed designs are able to speed up the AWM-Olsen application by 8% over the previous MPI-2 version, and 15% over the unmodified MPI-1 version.

The remainder of this paper is organized as follows. Section II briefly reviews AWM-Olsen, MPI-2 RMA semantics and its underlying implementation in MVAPICH2. In Section III, we provide a detailed performance analysis of AWM-Olsen and identify factors which may lead to reduced overlap in the MPI-2 version. Then in Section IV, we present our proposed enhancements. Our results are presented in Section V along with corresponding analysis. In Section VI we discuss the impact of our work on the real world usage and scaling of AWM-Olsen community model. Finally, we present the conclusions and future work in Section VII.

II. Background and Related Work

A. Anelastic Wave Model (AWM) and its Application

AWM-Olsen is a community model [4], [8], [9], [10], [11] used by researchers at the Southern California Earthquake Center (SCEC) for wave propagation simulations, dynamic fault rupture studies, physics-based seismic hazard analysis, and improvement of structural models. Some of the most detailed simulations to date of earthquakes along the San Andreas fault were carried out using this code, including the well-known TeraShake, SCEC ShakeOut simulations. The ShakeOut simulations, for example, modeled an M7.8 earthquake with $600\times300\times80$-km3 domain at 100-m resolution for up to four minutes. The most demanding simulation of a "wall-to-wall" rupture scenario required 32 billion volume elements and 86,000 time steps, ran for 20 hours on 32K processor cores of TACC Ranger, and generated 7 terabytes of surface output and checkpoints.

AWM-Olsen solves the 3D velocity-stress wave equation explicitly by a staggered-grid FD method, fourth-order accurate in space and 2nd-order accurate in time. The code includes a coarse-grained implementation of the memory variables for a constant-Q solid and Q relations validated against data. It uses Perfectly Matched Layers (PML) to implement absorbing boundary conditions on the sides and bottom of the grid, and a zero-stress free surface boundary condition at the top.

The velocity equations are:

$$\frac{\partial v_x}{\partial t} = \frac{1}{\rho}\left(\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + \frac{\partial \sigma_{xz}}{\partial z}\right)$$

$$\frac{\partial v_y}{\partial t} = \frac{1}{\rho}\left(\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{yz}}{\partial z}\right)$$

$$\frac{\partial v_z}{\partial t} = \frac{1}{\rho}\left(\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{yz}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z}\right)$$

where v_x, v_y and v_z are the x, y and z components of the velocity vector field, σ is the stress tensor, and ρ is density.

The stress equations are:

$$\frac{\partial \sigma_{xx}}{\partial t} = (\lambda + 2\mu)\frac{\partial v_x}{\partial x} + \lambda\left(\frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}\right)$$

$$\frac{\partial \sigma_{yy}}{\partial t} = \lambda\frac{\partial v_x}{\partial x} + (\lambda + 2\mu)\frac{\partial v_y}{\partial y} + \lambda\frac{\partial v_z}{\partial z}$$

$$\frac{\partial \sigma_{zz}}{\partial t} = \lambda\left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}\right) + (\lambda + 2\mu)\frac{\partial v_z}{\partial z}$$

$$\frac{\partial \sigma_{xy}}{\partial t} = \mu\left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x}\right)$$

$$\frac{\partial \sigma_{xz}}{\partial t} = \mu\left(\frac{\partial v_x}{\partial z} + \frac{\partial v_z}{\partial x}\right)$$

$$\frac{\partial \sigma_{yz}}{\partial t} = \mu\left(\frac{\partial v_y}{\partial z} + \frac{\partial v_z}{\partial y}\right)$$

where λ and μ are Lame's elastic constants.

The PML formulation for AWM-Olsen is described in detail by Marcinkovich and Olsen in [12]. Conventional wave propagation based on the equations above is carried out in the interior of the ground model. The regions on model sides and bottom require damping in their normal directions, so for example, the region at the minimum x boundary requires damping in x. The edges between boundaries require damping in the directions of both of the bounding planes and corners require damping in all three coordinate directions.

The 3D volume representing the ground area to be modeled is decomposed into 3D rectangular sub-grids. Each processor is responsible for performing stress and velocity calculations for its portion of the grid, as well as applying boundary conditions at the external edges of the volume if its sub-grid is on the boundary. Ghost cells, comprising a two-cell-thick padding layer, manage the most recently updated wave-field parameters exchanged from the edge of the neighboring sub-grids. Communication for the Ghost cell exchange can be overlapped with computation using either asynchronous 2-

sided send/receive calls or using MPI-2 one-sided communication as shown in [6]. Recent work by Cui, et al. [10], [11], enhanced the application through single-processor optimizations, optimization of I/O handling and optimization of TeraShake initialization. In this paper, we investigate obstacles to achieving full communication/computation overlap and how they can be overcome in this and other stencil-based codes.

B. Active Target Synchronization with MPI-2

Many network architectures, including InfiniBand [14], provide Remote Direct Memory Access (RDMA). Using RDMA, remote memory locations can be written or read without any involvement from the remote host processor. Therefore, RDMA provides opportunities for overlapping computation and communication. MPI-2 [15] introduced RMA semantics at the programming model level and enables applications to leverage the RDMA feature. It has since opened up new opportunities for parallel scientific applications. In order to use the RMA communication calls, each process group must define what is called as a *window*. Window creation is a collective call. Each process in the RMA access group contributes a local memory region to the group-wide window object. Remote processes can then access these regions via operations on the window. The RMA model decouples communication and synchronization, so it is non-blocking in nature and we can utilize it to achieve communication and computation overlap. Data transfer happens through communication calls. `MPI_Put` transfers data from the caller's (origin's) memory to the target's memory; `MPI_Get` transfers data from the target's memory to the caller's (origin's) memory; and `MPI_Accumulate` atomically updates the target's memory with the result of an operation on data from both the origin's and the target's memory. In these operations, all parameters required for the transfer are specified at the origin and thus intervention of the target in not required. Completion of these put, get and accumulate operations requires explicit synchronization calls. The MPI-2 RMA model defines two modes for synchronization: active and passive. Active synchronization involves both the origin and the target processes. It provides greater flexibility in synchronization by allowing the user to define sub MPI groups of a communicator and synchronize within this group. This flexibility is key for the use of Active Target Synchronization in AWM-Olsen. The main technique in using this mode for overlapping computation and communication is shown in Figure 1(b).

C. Underlying Implementation of MPI-2 RMA Communication Semantics

MPI-2 RMA semantics were introduced in the late 1990s. However, very few applications have been optimized to leverage RMA to the fullest extent. The reason for this is a cyclic dependency. MPI designers do not prioritize optimization of features that are not heavily used by end applications, at the same time, application developers tend to avoid features that are not very well optimized across a wide variety of MPI libraries. MVAPICH2 [7] is a popular MPI-2 library that has been specifically designed for InfiniBand and other RDMA capable networks. MVAPICH and MVAPICH2 are used worldwide (in 56 countries) by over 1,075 organizations. This software is distributed by major Linux distributors such as RedHat and SuSE. In addition, the software is available through OpenFabrics Enterprise Stack distributed by OpenFabrics Industry consortium [16]. Over the years, the MVAPICH/MVAPICH2 software stack has empowered many production InfiniBand clusters to obtain high ranking in the Top500 list; for example, top-ranking clusters from the most recent list (November 2009) using MVAPICH/MVAPICH2 include the 5th-ranked Tianhe-1 system [17] at NUDT in China with 71,680 cores; 9th-ranked Ranger system [5] at TACC with 62,976 cores; 27th-ranked Juno system [18] at LLNL with 18,224 cores; and 48th-ranked Chinook system [19] at PNNL with 18,176 cores.

MVAPICH2 has been specifically optimized for MPI-2 RMA semantics on InfiniBand, leveraging RDMA features [20], [21], [22]. Inter-node transfers initiated by `MPI_Put`, `MPI_Get` and `MPI_Accumulate` are carried out directly over RDMA for the Active Target synchronization semantics. For intra-node transfers (i.e. when origin and target processes are on the same physical node), the message transfer is done over a shared memory channel. The shared memory channel transfers messages using processor based copy operations. Although this provides fast transfers by avoiding network transactions, this consumes CPU cycles and thus, reduced overlapping capabilities. Recently, a new mode has been designed for intra-node transfers specifically for MPI-2 RMA communication in [23] which leverages Intel's I/O Acceleration Technology [24]. This exploits the presence of an on-board DMA engine to overlap memory transfers with computation. However, this depends on whether such a hardware feature is available on the target platform. We note that our target platform, Ranger cluster at TACC does not offer this feature.

III. Performance Analysis of AWM Olsen

In this section we provide a detailed analysis of the MPI-2 version of AWM-Olsen. We first analyze the computation routines and then computation aspects of the application. The focus of our analysis is to uncover behavior that may be reducing achieved overlap, as well as to detect dependencies that must be unraveled to increase chances of overlap.

A. Computation Performance Analysis

The main loop of the AWM Olsen consists of updating the velocity and stress variables for the sub-grid, applying PML boundary conditions for sub-grids containing boundary and exchanging ghost-cell data. The velocity values are updated for the interior and the boundary of the volume. Then velocity values are exchanged with processors containing the neighboring sub-grids in the directions of north, south, east, west, up and down as shown in Figure 2. This is followed by stress calculations and updates which are done in a similar manner.

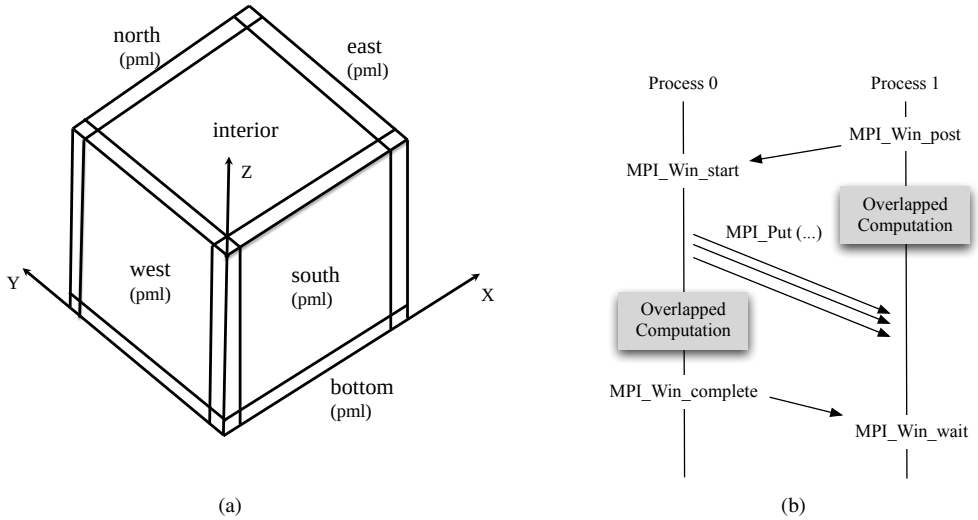


Fig. 1. (a) PML Boundary and Interior computation on a 3D Data Grid [13] (b) Overlap using MPI-2 RMA Target Active Synchronization Semantics

```
MPI_Win_post(group, 0, window) ! pre-posting the window to all neighbors
MAIN LOOP IN AWM-OLSEN
    ...
    Compute velocity component u
    Start exchanging velocity component u
    Compute velocity component v
    Start exchanging velocity component v
    Compute velocity component w
    Start exchanging velocity component w
    Complete Exchanges of u,v and w
    MPI_Win_post(group, 0, window) ! For the next iteration
    ...
START EXCHANGE
    MPI_Win_start(group, 0, window)
    s2n(u1,north-mpirank, south-mpirank) ! put to north window
    n2s(u1, south-mpirank, north-mpirank) ! put to south window
    ... repeat for east-west and up-down
COMPLETE EXCHANGE
    MPI_Win_complete(window)
    MPI_Win_wait(window)
    s2nfill(u1, window buffer, south-mpirank) ! ghost data from south
    n2sfill(u1, window buffer, north-mpirank) ! ghost data from north
    ... repeat for east-west and up-down

S2N
    Copy data of north face of subgrid for (u,v or w) variable to buffer
    MPI_Put(buffer, north-mpirank)

S2NFILL
    Copy data from window to (u,v, or w) variable south ghost cells
```

Fig. 2. Velocity update and ghost-cell exchange with 1-sided communication and Post-Wait/Start-Complete synchronization

1) Data Dependencies: The primary data values maintained for each grid point are the velocity vector field components: v_x, v_y and v_z and the stress tensor components: σ_{xx}, σ_{yy}, σ_{zz}, σ_{xy}, σ_{xz} and σ_{yz}, as given in the velocity and stress equations in Section II-A. From these equations it is clear to see that there are data dependencies between the velocity and stress variables. To meet these dependencies the velocity updates (including PML and sub-grid boundaries) must be complete before the stress calculations begin. Likewise the stress updates (including PML and sub-grid boundaries) must complete before the velocity updates for the subsequent time step of the main loop begin. The placement of communication initiation and completion are constrained by these dependencies.

2) Load Imbalance: At first glance this application looks to be well balanced, the grid is a structured rectangular grid split into equal sized rectangular sub-grids which are distributed among processors. However, the PML boundary conditions provide a source of imbalance. The computations corresponding to the equations for the PML boundary condition are applied to a 10 plane layer on the sides and bottom of the grid. Although the PML equations are similar to the interior equations, they can cause variations in computational load across processors in two ways: i) increased computational requirement per grid point, and ii) different data access patterns. The computation requirements of PML equations are higher, requiring more floating point operations. In addition, these equations are currently implemented in separate subroutines and loop nests from the main calculation. Further, the execution of PML equations is conditional (whether a particular block is on the boundary) and that the PML and interior field equations bounds are not compile time constants. This makes it very hard for compilers to automatically fuse portions of these loops. This leads to increased cache misses for the PML calculations and has a significant impact in creating a load imbalance amongst processors.

B. Communication Performance Analysis

In this section, we focus on the communication aspects of AWM-Olsen. We have further divided our analysis of communication based on inter-node communication and intra-node communication.

1) Inter-node Communication: The MPI-2 version of AWM-Olsen uses active target synchronization semantics. As mentioned in Section II-B, this synchronization mode allows synchronization in sub-groups. For example, in AWM-Olsen, processes need only to synchronize in their neighborhood, not

across all process in the communicator. This provides very low overhead synchronization. As mentioned in Section III-A1, the velocity vector fields must be computed before the stress tensor components. The key observation in [6] is that the computation of each component is independent of the other, and they can be overlapped. For example, transfer of v_x can be overlapped with computation of v_y, transfer of v_y can be overlapped with computation of v_w. However, the transfer of v_w cannot be overlapped, as it must finish before we start computation of stress tensor component σ_{xx}. Assume that T_{cv} is the time to transfer all components of velocity and T_{cs} is the time to transfer all components of stress. Since all velocity exchanges must complete before stress computation begins and all stress exchanges must complete before the next velocity computations begin, we can not overlap the transfer of the last component of both velocity and stress. Therefore, $1/3T_{cv}+1/6T_{cs}$ are not overlapped due to these dependencies.

2) Intra-node Communication: As mentioned in Section II-A, the AWM-Olsen application uses a 3D Cartesian grid of processes. The Ranger cluster has 16 cores on each node. AWM-Olsen assigns a linear mapping of contiguous ranks of the z dimension. Due to the node allocation strategy, contiguous ranks are placed on the same node in a "blocked" fashion. The processes communicate in a near-neighbor pattern. Therefore, two neighbors of each process are on the same node, leading to $1/3$rd of the total communication being intra-node.

MPI libraries often employ shared memory message passing techniques to implement communication within one node. Typical shared memory message passing schemes are implemented as follows: the origin processor P_1 copies message M into a shared memory buffer S. The target processor P_2 regularly polls queues in S to detect incoming messages. When an incoming message M is detected, it copies the message M to its destination receive buffer. Therefore, each message transfer typically incurs *two* memory copies. There are some mechanisms which can take help of a OS-privileged mode agent to reduce one copy, such as in [25]. However, there still exists one copy and the fact remains that the CPU is involved in message transfer and progress. As mentioned in Section II-C, a new I/OAT technology can be exploited to alleviate CPU involvement for intra-node message transfers. However, this feature is not widely deployed and is not accessible on our target platform, Ranger at TACC.

Yet another method to achieve intra-node message passing is to use the loop-back feature of the InfiniBand Host channel adapters (HCAs). The HCA can optimize message transfers between processes on the same node and which are connected via the same HCA. When such a situation is detected, the HCA simply DMAs the message from origin buffer to the target buffer. The message never leaves the node and all the communication is node-local. Typically, I/O bus bandwidth is lower than memory bandwidth, especially when a node architecture like Ranger is considered. There are 16 cores, all connected to a single HCA (SDR - 10Gigabits/s) through a single PCI-express (Gen1) bus. However, this method has a significant advantage, i.e. it does not require the involvement of the CPU to copy message from origin to destination buffer, leading to improved overlap.

In order to investigate the benefits of this approach, we designed a micro-benchmark to mimic the AWM-Olsen communication pattern. We extend the "ping-pong" benchmark to also include computation cycles. In our micro-benchmark, the origin process issues a `MPI_Put` followed by a pre-defined amount of computation. Then the origin process calls `MPI_Win_complete`. The target executes a `MPI_Win_wait` and replies back with `MPI_Put` of the same size of message it received. Meanwhile the origin is busy in the same pre-defined amount of computation. We see the results of this micro-benchmark in Figure 3(b) as compared to the normal ping-pong latency test. We observe that although the shared memory channel achieves best latency in the "no computation" scenario, network loop-back excels in the case of computation, adding very little overhead. Therefore, the network loop-back channel can completely overlap computation and communication.

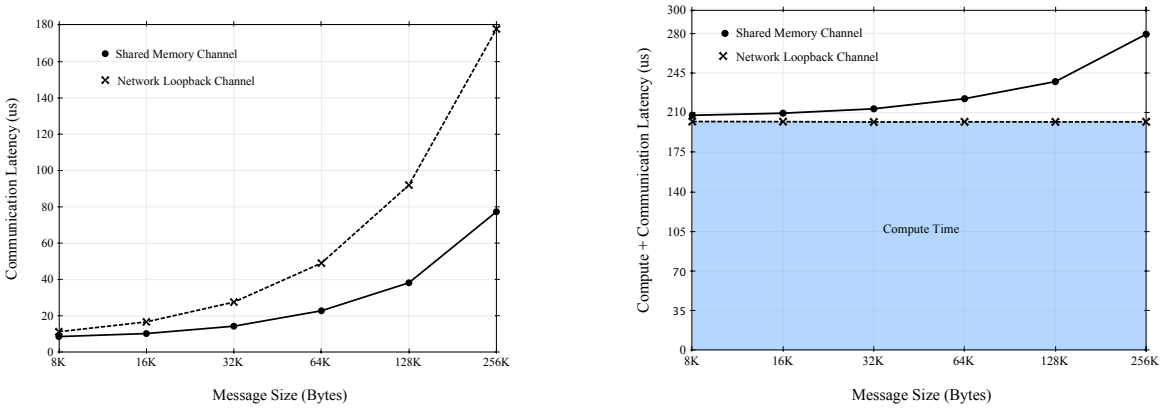
IV. Proposed Enhancements to AWM Olsen

A. Redesign using N, S, E, W, B boundaries

As presented in III-B1, the earlier design to separate and overlap the different components in velocity and stress has a limitation that the last components cannot be overlapped. This is due to the data dependencies explained in III-A1. Here, we alleviate this limitation by improving the earlier design to preferentially compute the boundary data of each component data grid and transfer it while computing the interior data grid of the same component.

Each component of velocity and stress corresponds to a 3-dimensional grid of data. In each communication step, every process exchanges the boundary data of these data grids with its neighbors in all directions. The computation of each component in velocity depends on the stress values computed in the previous iteration. A similar dependency exists for the computation of stress components. However, computation of different components within velocity/stress is independent of one-another. We had exploited this property in our earlier design. Similarly, the computation of different elements of a given component data grid is independent of one another. Using this behavior we can first compute the boundary data that is exchanged between the processes and then overlap this exchange with computation of the interior data. We can also achieve finer overlap by initiating the transfer of each boundary as it is computed and then move on to compute the other boundaries. For boundary processes, such a split is naturally allowed by the PML functions which compute the boundary planes. But for interior processes, this involves splitting the main computation loop.

Splitting the boundary and interior computation of a 3D data grid in this way, can have a significant impact on the cache behavior especially along the dimensions that result in unit-stride accesses in memory. For example, consider a system with a cache-line size of 64 bytes and a 2-way associative 64



(a) Ping-pong Latency

(b) Ping-pong Latency with Inserted Computation

Fig. 3. Performance of different Intra-node Communication Channels

KB L1 cache. Splitting the accesses to a 3D single-precision data grid, C(x=1:128,y=1:128,z=1:28), along the x dimension into boundaries : B1(1:2,1:128,1:28), B2(127:128,1:128,1:28) and interior I(3:126,1:128,1:28) can increase the L1 cache misses by 25 percent from 131,072 to 163,840. This will also incur cache misses at the higher level caches. To avoid this, we split the boundaries along the z and y axes and compute boundaries along the x axis along with the interior grid. We do not incur an increase in misses along the y and z axes as we still maintain the unit-stride access.

One disadvantage of this approach is that, exchange of boundaries along the x dimension, for the last component, cannot be overlapped with computation. However, we overlap these transfers with the copy-back (staging buffers to data grid) of completed transfers from the earlier components. [6] explains the staging buffers in more detail.

The positioning of window posts, completes and waits is crucial for achieving overlap with the Post-wait/Start-complete semantics. The windows should be posted in advance so that the they are available when the origin side issues the puts. The complete calls should be called as soon as the communication is expected to complete on the origin side and waits should be delayed as late as possible, placed just before the data is required for computation. This placement of the completes reduces wait time and ensures good overlap and delaying the waits helps in hiding process skew. Similar observations regarding best practices for placement of these calls are made in [26]. In the design in [6], all of the components (for example in velocity) are computed, exchanged and then completes are called for all the transfers. Waits immediately follow. Any process that is slow in reaching its complete calls will cause a delay in the wait calls of its neighbors. Therefore, in our new design, we issue the completes as soon as the transfers of each component are expected to complete, usually after a few boundaries of the next component are computed and their transfers are initiated. We delay the waits as long as possible as in our earlier version.

B. Reducing Load Imbalance using Loop Fusion

The new design to preferentially compute the boundaries is expected to improve communication hiding and thus reduce the time spent in `MPI_Win_complete`. However, we see that a notable amount of time is spent in `MPI_Win_wait` calls. Analysis in III-A2 shows there is a computation imbalance between the boundary and interior processes. For example, in the case of velocity, the boundary processes spend longer in computation as they need to do both the boundary PML computation and interior computation. In the contrary, for stress, the interior computation is more complex than the boundary computation and so the interior processes spend more time in computation. Though these two imbalances appear to get evened out, the data dependencies between velocity and stress require that some interior sub-grids wait for data from neighboring boundary sub-grids, delaying their start of the stress computation. Likewise, the boundary sub-grids will then have to wait for the stress results of neighboring interior sub-grids. These two add up to the time seen in `MPI_Win_wait`. Additionally, these delays propagate from sub-grid to sub-grid effectively limiting progress of all of the sub-grids. We believe, the solution to this is to individually reduce the computation imbalance in velocity and stress. In this work, we reduce the computation imbalance in velocity using the loop-fusion technique and demonstrate that this reduces the time seen in `MPI_Win_wait`.

As mentioned earlier, the main source for the computation imbalance in the current velocity computation design is that the boundary processes call two functions: $pmlvel$, to compute the boundary data and $dvel$, to compute the interior data, while the interior processes call just the $dvel$. The $pmlvel$ and $dvel$ functions access adjacent elements in each dimension of the data grid. These elements fall on the same cache-line in x dimension due to unit-stride access in memory. For example, the east boundary and interior in figure 1(a). However, as the east boundary and interior are computed by two independent functions, the boundary is fully computed before moving onto the interior. This results in duplicate misses of the shared

cache lines. Both $pmlvel$ and $dvel$ use all the data grids in stress while computing elements in velocity and this kind of access/miss pattern is true for all these grids.

Exploiting the match in access patterns, we fuse the loops from the two $pmlvel$ functions in the unit-stride dimension(x or east-west) with the loops of the $dvel$ function. Figure 4 shows an example for a process on the west boundary where the pml computation loop for component u is fused into its interior computation loop. This results in better cache behavior and lower computation time at the boundary processes. Such an optimization cannot be automatically done by a compiler as the loops exist in two different functions and the loop extents do not exactly match. The $pmlvel$ loops include update of the corner blocks which have to be separated out before the $pmlvel$ loops can be fused with the $dvel$ loops.

Existing computation of Velocity Component U at a Process on the West Boundary

```
PMLVELU
    for 10 k = 1,nz
    for 10 j = 1,ny
    for 10 i = 1,10
        compute u1(i,j,k)
    10 continue
DVELU
    for 20 k = 11,nz-10
    for 20 j = 11,ny-10
    for 20 i = 11,nx
        compute u1(i,j,k)
    20 continue
```

New fused-loop to Compute Velocity Component U at a Process on the West Boundary

```
PMLVELU_CORNERS
    for 30 k = 1,10
    for 30 j = 1,ny
    for 30 i = 1,10
        compute u1(i,j,k)
    30 continue
    for 40 k = nz-10,nz
    for 40 j = 1,ny
    for 40 i = 1,10
        compute u1(i,j,k)
    40 continue
    for 50 k = 11,nz-10
    for 50 j = 1,10
    for 50 i = 1,10
        compute u1(i,j,k)
    50 continue
    for 60 k = 11,nz-10
    for 60 j = ny-10,ny
    for 60 i = 1,10
        compute u1(i,j,k)
    60 continue
VELOCITYU_FUSED
    for 70 k = 11,nz-10
    for 70 j = 11,ny-10
    for 80 i = 1,10
        compute u1(i,j,k) using pml equations for boundaries
    80 continue
    for 90 i = 11,128
        compute u1(i,j,k) using equations for interior data
    90 continue
    70 continue
```

Fig. 4. Velocity computation using pmlvel and dvel functions and their fusion

V. Experimental Results

A. Experimental Setup

We have run all of our experiments on the TACC Ranger system. Ranger is a blade-based system. Each node is a SunBlade x6420 running a 2.6.18.8 Linux kernel. Each node contains four AMD Opteron Quad-Core 64-bit processors (16 cores in all) on a single board, as an SMP unit and has 32 GB of memory. The nodes are connected to an InfiniBand interconnect with Mellanox SDR adapters. In our experiments, we use MVAPICH2 1.4.1 as the underlying MPI implementation. We use a weak scaling model for our experiments to simulate real world application use. We increase the size of the data set as the process count increases such that data grid size per process remains at $128 \times 128 \times 128$ elements. Unless otherwise stated, the performance measurements presented are for 100 iterations of the main time-step loop in AWM-Olsen.

B. Evaluation

In the following sections, we refer to the existing version of the AWM-Olsen as AWM-MPI2-current. The version modified for preferential boundary computation, described in IV-A, is referred to as AWM-MCI2-ver1. AWM-MPI2-ver2 is the version re-designed with loop-fusion described in IV-A. The original MPI-1 based version is referred to as AWM-MPI1.

1) Minimizing Communication Overhead: In this section, we compare the communication performance of the current version of AWM-Olsen application with that of our new design for preferential boundary computation. AWM-MPI2-current is the existing version of the application as in [6] and AWM-MPI2-ver1 is the version with the optimization described in IV-A. In our experiments, we observed that using loop-back operations for communication in AWM-MPI2-current did not show any improvement in overlap because of the component-level overlap design. This confirms our understanding that the major part of the communication time is due to the non-overlapped last component and the computational skew which were not addressed in the earlier design. For all of the experiments, we have used shared-memory intra-node communication for AWM-MPI2-current and used loop-back communication for the new designs.

Figure 5(a) shows the average communication time while Figure 5(b) shows the communication time at the slowest process. Figure 5(c) shows the average communication time as a split between `MPI_Win_complete` and `MPI_Win_wait` times. We see that, at 8,192 Processes, the `MPI_Win_complete` time has reduced by close to 75% in AWM-MPI2-ver1 when compared to the current version. The gains here can be attributed to the finer grained overlap of the last components in velocity and stress, achieved by preferentially computing the boundaries and the use of loop-back operations for intra-node communication. At the same process count, the `MPI_Win_wait` times have been reduced by 64%. Posting the complete calls immediately after the transfers are expected to finish helped hide part of the computation skew between processes, thus resulting in the reduction

in wait times we observe here. The average communication time of the application has reduced by 67% and the maximum communication time at any process has reduced by 47%. We performed four runs for each of the experiments and we observed only a 0.2% deviation in the numbers which shows the consistency in gains. Figure 5(d) depicts these numbers.

As mentioned earlier, our main experimental platform, TACC Ranger, does not have I/OAT like features to enable asynchronous communication progress of intra-node communication. However, to confirm that our designs can be effectively use such features when available, we ran 64 process (8 processes per node) experiments on a smaller Intel Clovertown cluster enabled with the I/OAT feature. We use the intra-node communication design for MPI-2 RMA presented by Ping, et al. in [23]. Figure 9(b) shows that we achieve similar improvements in performance due to overlap using I/OAT and loop-back techniques.

2) Reducing Computation Imbalance: Figure 7(a) shows the skew in velocity computation between processes for a 4K run using AWM-MPI2-current and figure 7(b) shows the skew after loop-fusion has been applied, in AWM-MPI2-ver2. We see that the skew has been reduced by 39% from 5.42 secs to 3.26 secs. We have used cachegrind to confirm that the new design using loop-fusion results in a better cache behavior than the earlier design. We observed a 14% reduction in L1 read misses for velocity computation and an 9% reduction in L2 Read misses. The L1 and L2 write misses have dropped by 33%. These results are summarized in Figure 6. We also see a corresponding reduction in MPI_Win_wait times in figure 5(c). The overall average communication time has improved by 15%, when compared to AWM-MPI2-ver1 and by 72.5%, when compared to AWM-MPI2-current on 8K processes.

3) Application run-time: Figures 8(a) and 8(b) compare the total run-time of the current version of AWM-Olsen with the two new versions. We see that AWM-MPI2-ver1 runs faster by 4% and AWM-MPI2-ver2 runs faster by 8%, compared to the current version on 8K cores. Figure 8(d) shows that the performance gains remain stable as we scale from 1K to 8K processes.

VI. Impact on Field of Research

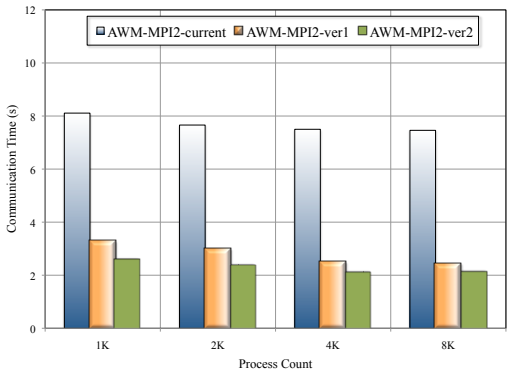
Next-generation exascale systems are expected to provide a massive increase in aggregate compute power (FLOPS). Inherent communication costs in the underlying parallel algorithms usually prevent perfect scaling. In order to scale to next generation systems, it is extremely critical that communication costs be minimized to the extent possible. Efficient communication design is the key to achieve such scalability. For example, the original AWM-Olsen application written using MPI-1 blocking mode communication calls spends around 24% of its execution time in `MPI_Waitall` calls (at the slowest process) for data grid size of $128 \times 128 \times 128$ at 8K process count. Such large times spent in communication routines are unacceptable on very large scale system. This leads to waste of valuable system time. Many of the upcoming systems require application to be able to exploit features like RDMA and have designs for computation-communication overlap. Our work enables AWM-Olsen to meet this requirement. Our MPI-2 RMA based designs proposed in [6] reduce the average and maximum times spent in communication progress to 7% and 20% respectively. Work presented in this paper enhances these designs further to reduce the times to 2% and 8.9% respectively. Total communication time has been reduced by 72% over the previous MPI-2 version and by 89% over the unmodified MPI-1 version on 8K processes. Figure 9(a) shows that our best version improves the application run-time by 15% when compared to the original application.

It is expected that in the near future, nodes will contain many cores, may be up to 64 or 128. Therefore, "fatter" nodes will become common and efficient intra-node communication will become more important. Typical shared memory implementations in MPI burn CPU cycles for memory copies. This results in reduced overlap. On the other hand, technologies like I/OAT [24] are emerging with a promise of offloading data-transfer duties to on-board DMA engines. Ping, et al. have leveraged I/OAT and designed truly one-sided communication primitives for MPI-2 RMA in [23]. Although our target cluster, Ranger, does not have this hardware feature, we explored the viability of I/OAT for AWM-Olsen. We ran the various versions of AWM-Olsen on a smaller Intel Clovertown cluster which has hardware I/OAT. The results are shown in Figure 9(b). We observe that the schemes developed in this paper (AWM-MPI2-ver2), is able to achieve best performance with both I/OAT and Network loop-back. This demonstrates the effectiveness of our optimizations and the I/OAT feature.

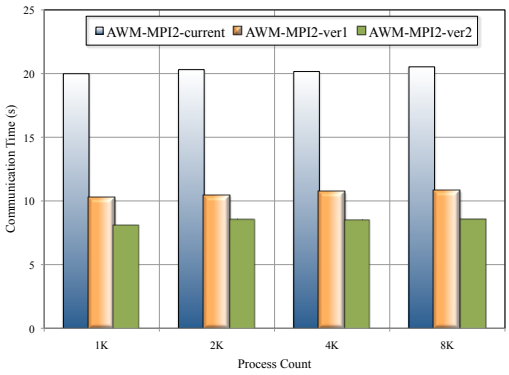
As mentioned in II-C, not many state-of-the-art MPI libraries have efficient implementations for MPI-2 RMA semantics. Lack of applications that demand these features is a key reason for this lapse. On the other hand, application developers have little incentive and literature to their aid, on how to efficiently use these semantics for their applications. Lack of good implementations is also a hindrance, making the dependency reciprocal. Through our work, we hope to influence and help more application developers to effectively use MPI-2 RMA semantics to enhance their applications with MPI-2 RMA based designs for better communication-computation overlap.

VII. Conclusion and Future work

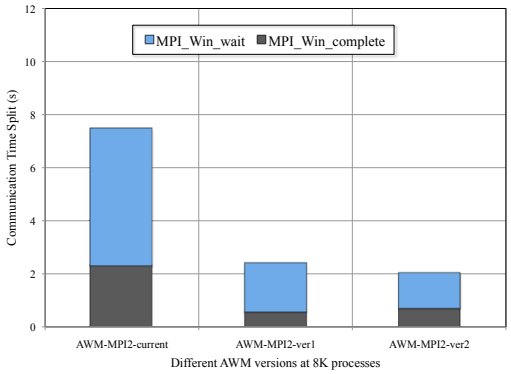
AWM-Olsen is a widely used earthquake-induced ground wave-propagation simulation code which consumes several million CPU hours every year on the TeraGrid clusters. Efficient communication design is paramount for such large scale, heavily used applications to best utilize the available system time and resources. For the same reason, modern systems require applications to have designs that exploit features like RDMA for communication-computation overlap. Our work in [6] had modified the communication design in AWM-Olsen to use MPI-2 RMA semantics. In this paper, we first analyzed the limitations of the earlier design and enhanced it further to significantly reduce the time spent in communication progress. Using the combination of our proposed optimizations, time



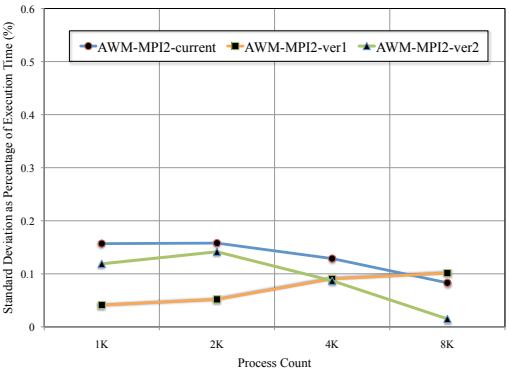
(a) Average communication time across all processes



(b) Maximum communication time at slowest process



(c) Communication cost as a sum of its parts at 8K processes

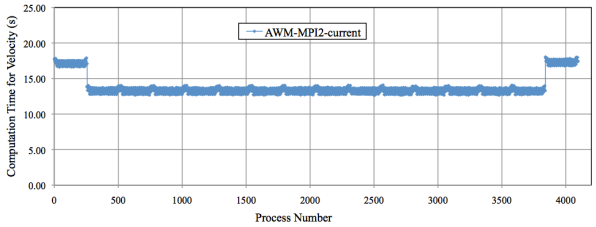


(d) Standard deviation of performance data

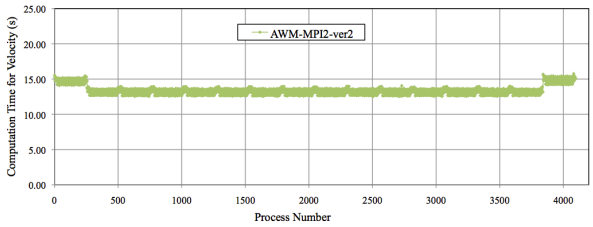
Fig. 5. Performance improvement and reduction in communication time for AWM-Olsen at scale

Version	Reads	L1 Read Misses	L2 Read Misses	Writes	L1 Write Misses	L2 Write misses
AWM-MPI2-current	58,362,427	4,016,665	3,449,979	4,783,794	213,070	211,203
AWM-MPI2-ver2	51,748,613	3,454,385	3,140,644	4,544,882	143,264	140,586

Fig. 6. Cache Analysis (per iteration) using Valgrind on Velocity Compute Functions with and without Loop-Fusion



(a) Skew Regular



(b) Skew Loop Fused

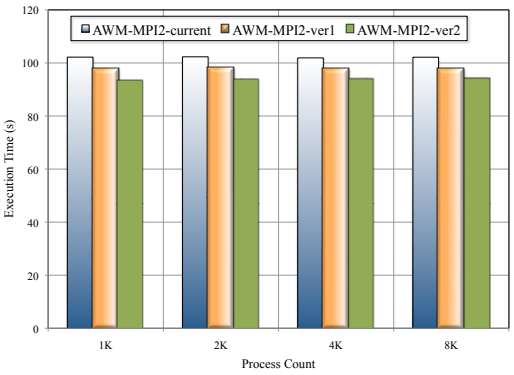
Fig. 7. Impact of Reduction of Process Skew using Loop Fusion

spent in communication can be reduced by 72% over the previous MPI-2 version and by 89% over the unmodified MPI-1 version on 8K processes. This results in an overall speedup in the execution time by 8% and 15% respectively.

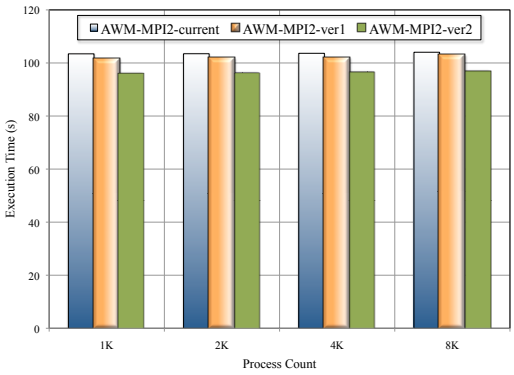
In the near future, we would like to extend this work to achieve near-perfect computation balance using different load-balancing techniques. We would like to explore the angle of developing parallel programming patterns which can be used by application writers to achieve similar overlap benefits without requiring a deep understanding of the MPI-2 one-sided semantics.

VIII. Acknowledgements

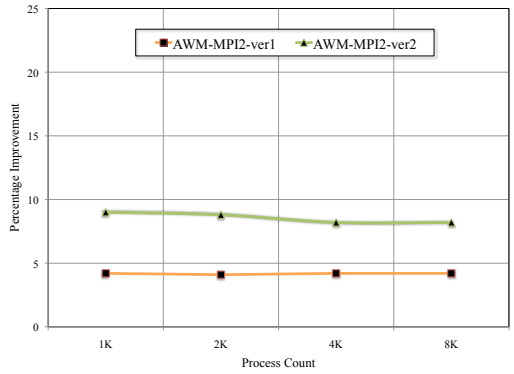
The authors wish to thank Karl Schulz and William Barth from Texas Advanced Computing Center (TACC) at The University of Texas at Austin, for their support and for providing HPC resources on the Ranger that have contributed to the research results reported within this paper. This research is supported in part by DOE grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342, #CCF-0702675, #CCF-0833169, CCF-0833139, CCF-0833155, #CCF-0916302 and #OCI-0926691; grants from Intel, Mellanox, Cisco systems, QLogic and Sun Microsystems; and equipment donations from Intel, Mellanox, AMD, Appro, Chelsio, Dell, Fujitsu, Fulcrum, Microway,



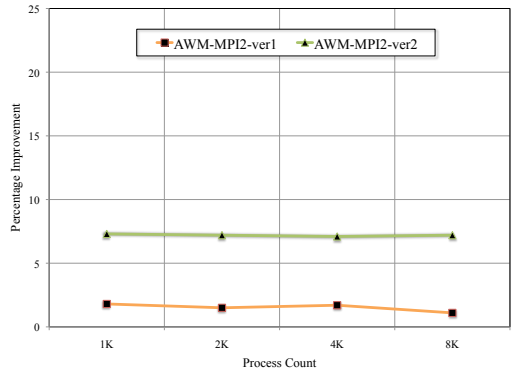
(a) Average execution time across all processes



(b) Maximum execution time at slowest process

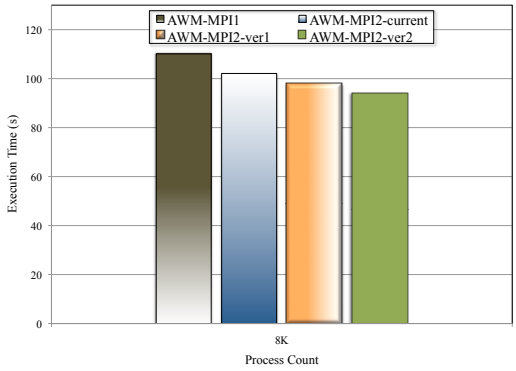


(c) Percentage improvement in average execution time

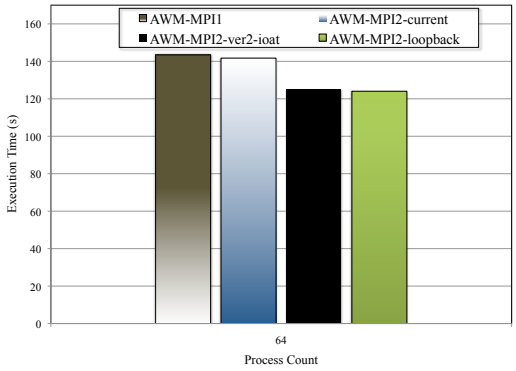


(d) Percentage improvement in maximum execution time

Fig. 8. Performance improvement and reduction in execution time for AWM-Olsen at scale



(a) Overall improvement of AWM-Olsen across various versions at scale



(b) Effectiveness of I/O Acceleration Mechanism

Fig. 9. Performance improvement of AWM-Olsen over original design

Obsidian, QLogic, and Sun Microsystems.

REFERENCES

[1] National Center for Computational Sciences (NCCS). Jaguar Supercomputer. [Online]. Available: http://www.nccs.gov/computing-resources/jaguar/

[2] National Center for Supercomputing Applications (NCSA). Blue Waters, Sustained Petascale Computing. [Online]. Available: http://www.ncsa.illinois.edu/BlueWaters/

[3] R. Fiedler and D. S. Katz and W. Kramer. Early Access to the Blue Waters Sustained Petascale System. [Online]. Available: http://www.ncsa.illinois.edu/BlueWaters/pdfs/webinar_Prospective_PRAC.pdf

[4] K. B. Olsen, "Simulation of three-dimensional wave propagation in the Salt Lake Basin," University of Utah, Salt Lake City, Utah, Tech. Rep., 1994.

[5] Texas Advanced Computing Center. Ranger Cluster. [Online]. Available: http://www.tacc.utexas.edu/resources/hpc/

[6] S. Potluri, P. Lai, K. Tomko, S. Sur, Y. Cui, M. Tatineni, K. Schulz, W. Barth, A. Majumdar, and D. K. Panda, "Quantifying Performance Benefits of Overlap using MPI-2 in a Seismic Modeling Application," in *International Conference on Supercomputing (ICS'10)*, 2010. [Online]. Available: http://nowlab.cse.ohio-state.edu/publications/conf-papers/2010/potluri_ics2010.pdf

[7] Network-Based Computing Laboratory. MVAPICH/MVAPICH2: MPI over InfiniBand, 10GigE/iWARP and RDMAoE. [Online]. Available: http://mvapich.cse.ohio-state.edu

[8] K. Olsen, S. Day, J. Minster, Y. Cui, A. Chourasia, M. Faerman, R. Moore, P. Maechling, and T. Jordan, "Strong Shaking in Los Angeles Expected from Southern San Andreas Earthquake," in *Geophysical Research Letters, Vol 3*, 2006, pp. 1–4.

[9] Y. Cui, K. B. Olsen, Y. Hu, S. Day, L. A. Dalguer, B. Minster, R. Moore, J. Zhu, P. Maechling, and T. Jordan, "Optimization and Scalability of a Large-scale Earthquake Simulation Application," in *American Geophysical Union, Fall Meeting*, 2006.

[10] Y. Cui, R. Moore, K. Olsen, A. Chourasia, P. Maechling, B. Minster, S. Day, Y. Hu, J. Zhu, A. Majumdar, and T. Jordan, "Enabling Very-Large Scale Earthquake Simulations on Parallel Machines," in *ICCS '07: Proceedings of the 7th international conference on Computational Science, Part I*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 46–53.

[11] Y. Cui, R. Moore, K. Olsen, A. Chourasia, P. Maechling, B. Minster, S. Day, Y. Hu, J. Zhu, and T. Jordan, "Toward Petascale Earthquake Simulations," in *Acta Geotechnica*, 2009, pp. 79–93.

[12] C. Marcinkovich and K. Olsen, "On the implementation of perfectly matched layers in a three-dimensional fourth-order velocity-stress finite difference scheme," *J. Geophys. Res. 108(B5)*, 2003.

[13] ——, "On the implementation of perfectly matched layers in a three-dimensional fourth-order velocity-stress finite difference scheme," in *Geophysical Research Letters, 108(B5)*, 2003.

[14] Infiniband Trade Association. InfiniBand Network Architecture. [Online]. Available: http://www.infinibandta.org

[15] *MPI-2: Extensions to the Message-Passing Interface*, Message Passing Interface Forum, Jul 1997.

[16] The Open Fabrics Alliance. The Open Fabrics Enterprise Distribution. [Online]. Available: http://www.openfabrics.org/

[17] National SuperComputer Center in Tianjin/NUDT. Tianhe-1. [Online]. Available: http://www.nscc-tj.gov.cn/en/

[18] Lawrence Livermore National Laboratory. Juno. [Online]. Available: https://computing.llnl.gov/?set=resources&page=SCF_resources#juno

[19] Pacific Northwest National Laboratory. Chinook. [Online]. Available: http://www.pnl.gov/

[20] W. Jiang, J. Liu, H.-W. Jin, D. K. Panda, W. Gropp, and R. Thakur, "High performance MPI-2 One-sided Communication over InfiniBand," in *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 531–538.

[21] W. Huang, G. Santhanaraman, H.-W. Jin, and D. K. Panda, "Scheduling of MPI-2 One Sided Operations over InfiniBand," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 9*. Washington, DC, USA: IEEE Computer Society, 2005, p. 215.1.

[22] G. Santhanaraman, P. Balaji, K. Gopalakrishnan, R. Thakur, W. Gropp, and D. K. Panda, "Natively Supporting True One-Sided Communication in MPI on Multi-core Systems with InfiniBand," in *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 380–387.

[23] P. Lai, S. Sur and D. K. Panda, "Designing Truly One-Sided MPI-2 RMA Intra-node Communication on Multi-core Systems," in *International Supercomputing Conference (ISC 2010)*, June 2010.

[24] Intel Corp. I/O Acceleration Technology. [Online]. Available: http://www.intel.com/network/connectivity/vtc_ioat.htm

[25] H. -W Jin and S. Sur and D. K. Panda, "LiMIC: Support for High-Performance MPI Intra-Node Communication on Linux Cluster," in *In Proceedings of the 2005 International Conference on Parallel Processing (ICPP)*, 2005.

[26] M.-A. Hermanns, M. Geimer, B. Mohr, and F. Wolf, "Scalable detection of MPI-2 remote memory access inefficiency patterns," in *16th Euro PVM/MPI, Lecture Notes in Computer Science*. Springer-Verlag, 2009.